# Parallel Computation of Second Derivatives of RHF Energy on Distributed Memory Computers

**ANTONIO M. MÁRQUEZ,* JAIME OVIEDO, and JAVIER FERNÁNDEZ SANZ**

*Departamento de Química Física, Facultad de Química, C/Prof. García González s/n, Sevilla, E41012, Spain*

**MICHEL DUPUIS†**

*IBM Corporation, Department MLMA/MS 428, Neighborhood Road, Kingston, New York 12401*

*Received 16 June 1995; accepted 3 May 1996*

**ABSTRACT**

A parallel implementation of the computation of RHF energy second derivatives with respect to the nuclear coordinates is described. The algorithm and organization of the code are described in detail on the most computationally demanding steps with special emphasis on the integral transformation code. Key features of the proposed algorithm are its large degree of concurrency, limited interprocessor communication, and critical memory needs distributed over the processors. The cpu times and computer and network resources used are reported and discussed for a few examples. © 1997 by John Wiley & Sons, Inc.

## Introduction

Analytical derivative methods have been developing continuously in quantum chemistry since their introduction in the late 1960s.[1] A wide range of problems in quantum chemistry require the computation of derivatives of energy with respect to certain parameters.

The total energy of a molecular system as a function of the nuclear coordinates is called the potential energy (hyper)surface. The shape of this surface can be directly related to the behavior of the molecular system as a function of the nuclear coordinates. Derivatives of the energy with respect to the nuclear coordinates can also be correlated to physical observables. Thus, the negative of the first derivative vector represents the force exerted on the nucleus by the presence of the other nuclei and

* Author to whom all correspondence should be addressed. E-mail: marquez@quantix.us.es

† Present address: Batelle, Pacific Nothwest Laboratory, P.O. Box 999, Richland, WA 99352.



CCC 0192-8651/97/020159-10

electrons and bears a direct relation to the location of stationary points on the molecular energy hypersurface. The second derivatives are the force constants; they determine the harmonic vibrational spectrum of the molecule. Cubic and higher order derivatives can be related to anharmonic effects on the vibrational spectrum.

In recent years it has become increasingly possible to study ''real,'' highly complex molecule systems. Quantum chemistry methods exhibit a large nonlinear scaling of computational requirements with the size of the systems simulated, and their successful application requires a balanced mix of raw computing power and theoretical and computational innovation. Parallel computers are opening the door to the study of increasingly large molecular systems, but the adequate exploitation of this new computer architecture requires fundamental algorithmic innovation to make a real impact on the study of such large molecular systems.[2] Parallelization of *ab initio* codes is a very active area of research with many new ideas continuously emerging. Most of the work has been directed toward the parallelization of the two-electron repulsion integrals, gradient and SCF codes,[3] integral transformation,[4] MP2 energy,[5] configuration interaction energy,[4, 6] and coupled cluster programs.[7]

Distributed computing is emerging as a major development affecting scientific problem solving. Distributed computing is a parallel computing design in which a set of computers is used collectively to solve a single large problem. Common to all parallel processing is the exchange of data between cooperating tasks. In distributed computing, message passing has become the paradigm of choice in terms of applications, languages, and software systems that use it. Between the number of packages that allow for a portable message-passing parallel coding, we have chosen PVM[8] as our parallel programming environment, although our implementation can be easily ported to other packages that support a message-passing model.

In this article, we will describe the parallelization of the RHF analytical Hessian module of the HONDO program system using the PVM programming environment and its performance with representative examples. We then review the basic theory of the problem. Next, we describe the parallel algorithm implemented with special emphasis in the integral transformation code. The subsequent section examines the expected performance of the program in terms of a model in which we review important issues such as parallel execution time, overhead sequential time, and communication time in terms of problem size and the processor count. The final section presents results on representative examples, discussing the observed performance.

## Theory

For our present purposes we will give a brief review of the basic equations needed for the computation of the second derivatives of the RHF energy. For a detailed discussion we refer the reader to refs. 1 and 9.

The general expression for the second derivatives of the energy with respect to two nonvariational parameters is:

$$E^{ab} = W^{ab} + \sum_x W_x^a \tilde{x}^b = W^{ab} + \sum_x W_x^b \tilde{x}^a \quad (1)$$

In this expression, $W$ is the trial energy computed for given nonvariational parameters $(a, b, c, \ldots)$ and for given variational parameters $(x, y, \ldots)$, and $E$ refers to the energy computed for the optimal variational parameters $(\tilde{x}, \tilde{y}, \ldots)$. The superindices refer to differentiation with respect to perturbational parameters and subindices with respect to the variational parameters. We adopt the following notation for the orbitals: $i, j, k, l$ refer to occupied molecular orbitals (MO) and $u, v$ refer to virtual MO; $p, q, r, s$ refer to atomic orbitals (AO).

Taking the expression for the RHF energy and differentiating twice with respect to nuclear coordinates, one obtains the following expression in terms of molecular orbitals for the $W^{ab}$ matrix:

$$W^{ab} = \tilde{W}^{ab} + \sum_{ij}^{occ} T_{ij}^a A_{ij}^b + T_{ij}^b A_{ij}^a - 2\sum_i^{occ} \epsilon_i S_{ii}^{ab} + 8\sum_{ij}^{occ} (\epsilon_i + \epsilon_j) T_{ij}^a T_{ij}^b \quad (2)$$

The $\tilde{W}^{ab}$ term has the form of the energy expression evaluated using second derivative integrals. The **A** matrix is defined as:

$$A_{ij}^a = 4\tilde{F}_{ij}^a + 2\sum_{kl}^{occ} [4(ij \mid kl) - (ik \mid jl) - (il \mid jk)] T_{kl}^a \quad (3)$$

where $\tilde{F}^a$ is a Fock-like matrix constructed using first derivative integrals. The $W_x^a$ matrix corresponding to a given orbital rotation, $x_{iu}$, is defined

to be as:[10]

$$W_x^a = 8\epsilon_i T_{iu}^a + 4\tilde{F}_{iu}^a + 4\sum_{jk}^{occ}[4(iu \mid jk) - (ik \mid ju) - (ij \mid ku)]T_{jk}^a \quad (4)$$

The derivatives of the orbital rotations, $\tilde{x}^a$, are obtained by the solution of the so-called coupled perturbed Hartree–Fock (CPHF) equations:

$$\sum_y W_{xy}\tilde{y}^a = -W_x^a \quad (5)$$

where the $W_{xy}$ matrix element is defined as:

$$W_{xy} = 4\delta_{ij}\delta_{uv}(\epsilon_u - \epsilon_i) + 16(iu \mid jv) - 4(ij \mid uv) - 4(iv \mid ju) \quad (6)$$

for the pair of orbital rotations $x_{iu}$ and $y_{jv}$.

In eqs. (1)–(6), $\epsilon_i$ are the one-electron MO energies and the **T** matrix defines the effect of the perturbational parameters $(a, b, \ldots)$ in the MO coefficients.[9] First and higher order derivatives of the **T** matrix are easily computed and its derivatives with respect to the variational parameters are zero.

## Program Organization

The general organization of our parallel SCF code has been previously discussed and only a brief summary will be given here.[5b] The program is organized following the SPMD (single program multiple data) paradigm. The user starts a *master* program that determines, by reading the input file, the number of threads to run $(P)$ and starts $P-1$ copies of itself (nodes) on the parallel machine. The input file is read by the master and sent to the nodes. In this way all processes have all necessary information about the molecule to start the calculation. At this point, all threads begin computation separately. They perform the necessary set-up of data (initial guess orbitals, one-electron integrals, etc.) and compute independently their share of the two-electron integrals. After this task is completed, each thread computes an incomplete Fock matrix by contracting its sublist of the two-electron integrals with the density matrix. Then a global add operation is performed to add all contributions together and form the complete Fock matrix. The SCF energy is then computed, convergency is checked, and a new SCF iteration begins if necessary.

After evaluation of the SCF wave function, the next task is the computation of the first and second derivatives of the one- and two-electron integrals. Computation of the one-electron integrals derivatives does not constitute a significant part of the work and no effort has been made to parallelize this step. Computation of the two-electron integrals derivatives is a very cpu-intensive task and it accounts for a significant percentage of the total work.[9a] As in the evaluation of the two-electron integrals, the program is parallelized by distributing the loop over shells over the available processors. This technique produces an excellent load balancing[3b] and is scalable up to a large number of processors.[3m] The first- and second-derivative integrals are not stored but rather used as soon as they are generated. The second derivative integrals are multiplied by two-particle density matrix elements (also generated on the fly) and its contribution added to the $\tilde{W}^{ab}$ matrix which is kept in memory. In a similar way, first-derivative integrals are contracted with one-particle density matrix elements and added into the derivative Fock operators, $\tilde{F}^a$, which are also kept in main memory.

The next task is the transformation of the two-electron integrals, needed for the resolution of the CPHF equations. The transformation of the two-electron integrals is done without a presorting step; that is, the unordered list of AO integrals is taken and the first index transformation is done as follows:

$$(pq \mid rj) = \sum_s^N C_{sj}(pq \mid rs) \qquad j = 1, n_{occ} \quad (7)$$

where $N$ is the number of basis functions and $n_{occ}$ is the number of occupied MO. The process is highly cpu intensive as well as memory intensive and, in most cases, I/O intensive as well. It is important to consider the memory requirements to perform this step. To hold all $(pq \mid rj)$ in memory, $n_{occ}N^2(N+1)/2$ words of memory are needed as the $pq$ indices are treated as a combined index. If less memory is available, the program proceeds by batches, transforming as many occupied orbitals $(j)$ as possible for the available memory. The minimum memory required is, thus, $N^2(N+1)/2$ words if only one MO can be transformed on each batch. This approach is similar to the one described by Almlöf et al.,[11] except that we prefer to

complete the first index transformation before transforming the rest of the indices. This requires extra core storage with respect to their approach but allows us to use fast matrix–matrix multiplication routines on those steps.

When working in parallel the AO integrals are distributed over the different processors which is an important point in the design of the integral transformation program. Because of the fourth rank tensor nature of the integrals, the computation of a given $(pq \mid rj)$ requires AO integrals $(pq \mid rs)$ with all possible values of $s$ that can be distributed over different processors. In a previous work,[5b] parallelization of the two-electron integral transformation step was done by distributing the loop over occupied orbitals in eq. (7) over the available processors. However, this requires that each thread transforms its own sublist of AO integrals, as well as those from other threads, thus each thread needs to process the entire list of AO integrals. This was done by using a broadcast function: each thread, in turn, takes a buffer of its own sublist of AO integrals, broadcasts this buffer to the other processors, and reads one buffer from each one of them. A simple analysis of the amount of data transferred between the processors shows that, because of this broadcasting of data, it grows proportionally with the number of processors. This means that, although this algorithm is applicable to parallel computers with very few processors, it cannot be scaled to massively parallel computers composed of many processors.

In this program, parallelization of eq. (7) is done by distributing the combined index $pq$ between the available processors. Each thread takes a buffer of AO integrals and classifies them according to their $pq$ index in buckets for the different processors. Then, in a collective operation, all threads exchange buckets so that, after his operation, each thread has a set of AO integrals to be processed according to eq. (7) to form a subset of the one-index-transformed integrals. The process is repeated until all threads have processed all AO integrals. By performing this index distribution, memory requirements are also distributed between processors: the minimum global memory needed to transform one occupied molecular orbital is still $N^2(N+1)/2$, but the $pq$ indices are now distributed between processors and the memory needed on each of them is reduced to $N^2(N+1)/(2 \cdot p)$.

After this first transformation has been accomplished, the rest of the indices are transformed stepwise to produce the final set of MO integrals. First, the $r$-index is transformed according to:

$$(pq \mid tj) = \sum_{r}^{N} C_{rt}(pq \mid rj) \tag{8}$$

where $t$ refers to both occupied and virtual MO. As every thread has all $r$-indices, this transformation can be performed without further communication between threads. For the transformation of the $pq$ indices it is necessary to redistribute the integrals between the threads. Now the $tj$ indices are distributed over the processors and it is necessary for each thread to have all $pq$ indices for a given $tj$ index. In a similar way as in the first index transformation, each thread puts the appropriate $pq$ indices in buffers and sends them to their corresponding processors. When all processors have their buffers full, they transform the $pq$ indices in parallel, resulting in the final set of MO integrals that are stored on disk for later use.

Figure 1 shows schematically the structure of computation on the integral transformation step. An important point in this algorithm is that all communication is established on a point-to-point basis, never using the equivalent of broadcast functions. This implies that the amount of communication is kept constant, or at least, has a finite limit when the number of processors increases, resulting in an algorithm that is suitable for use in parallel computers with many processors. This statement can be rationalized as follows: Let us say that the total number of two-electron repulsion integrals is $\tau$; thus, each processor has $\tau/P$ AO integrals. On the first index integral transformation each processor will distribute this subset of integrals between all processors (including itself), resulting in $\tau(P-1)/P^2$ integrals being written to the network per processor and a total network traffic of $\tau(P-1)/P$ integrals. For the redistribution of the integrals prior to the transformation of the $pq$ indices the argument is similar and also results in a $O((P-1)/P)$ dependency of network traffic with the number of processors.

Table I shows an interesting comparison of the integral transformation algorithms employed in ref. 5b and in this work. The grain size, defined as the minimum amount of work a processor can be given, has been decreased by an order of magnitude, resulting in a much better processor load balancing. The amount of communication between the processors does not now grow proportionally with the number of processors, resulting in a lim-

```
loop over batches of j

loop over (pq|rs) integrals
        sort them in buckets - pq
        exchange buckets
        form (pq|rj) += C_sj (pq|rs) for all buckets
end loop
form (p̄q̄|tj) += C_rt (p̄q̄|rj)
loop over tj indices
        if mod(tj,nap)+1 = iap then
                receive (pq|tj) integrals from other threads
                form final MO integrals
        else
                send (pq|tj) integrals to its thread
        end if
end loop

end loop over batches of j
```

**FIGURE 1.** Structure of integral transformation process.

**TABLE I.**
**Comparison of Integral Transformation Algorithms.[a]**

| | Ref. 5b | This work |
|---|---|---|
| Grain size | $O(N^3)$ | $O(N^2)$ |
| Concurrency | $n$ | $N^2$ |
| Interprocessor communication | $O(P)$ | $O((P-1)/P)$ |
| Minimum memory (per node) | $N^2(N+1)/2$ | $N^2(N+1)/(2P)$ |

[a] $N$ refers to the number of atomic orbitals, $n$ to the number of occupied molecular orbitals, and $P$ to the number of processors.

ited interprocessor data transfer. Also, the minimum memory needed to perform the calculation has been distributed over the processors thus extending the range of applicability of the program to larger molecular systems.

Once the computation of the two-electron MO integrals is achieved they are used in a first step to complete the $A^a$ matrices [eq. (3)] and the $W_x^a$ matrices [eq. (4)]. As the MO integrals are distributed over the processors, this step is parallelized by having each processor compute one incomplete contribution to those matrices from its set of MO integrals. In the end, all contributions are added together (in a global add operation) and added to the $A^a$ and $W_x^a$ matrices.

The set of CPHF equations is then solved by a subspace expansion method in which expansion vectors are obtained iteratively by a first-order correction procedure.[12] This procedure usually requires only a few iterations to converge. The CPHF equations are solved simultaneously for the displacements of symmetry-unique nuclei and the solutions are mapped to the symmetry-equivalent nuclei.

In our implementation, the $W_{xy}$ matrix is never built in memory; instead, MO integrals are read from disk at each iteration to directly compute the product of the current expansion vectors, $\boldsymbol{\alpha}^a$, by the $W_{xy}$ matrix and form the new set of correction vectors:

$$\begin{aligned}\boldsymbol{\sigma}_x^a &= \sum_y W_{xy} \cdot \boldsymbol{\alpha}_y^a \\ &= \sum_{jv} \left[ 4\delta_{ij}\delta_{uv}(\epsilon_u - \epsilon_i) + 16(iu \mid jv) \right. \\ &\quad \left. - 4(ij \mid uv) - 4(iv \mid ju) \right] \cdot \boldsymbol{\alpha}_{jv}^a\end{aligned}$$

This step is easily parallelized by having each processor compute independently one set of incomplete correction vectors from its own sublist of MO integrals. The complete set of correction vectors is then obtained by adding the contributions from all threads in a global add operation.

## The Model

In parallel programming it is quite useful to have a quantitative model of the performance of the parallel code with varying problem sizes and processor counts. Ideally, the models should not be limited to a single metric such as cpu time; rather, a good model should include different pa-

rameters such as execution time, memory requirements, disk and network I/O, and possibly other factors. The model for a specific algorithm can be used to evaluate its scalability, to identify bottlenecks, and to compare its efficiency with that of other algorithms.

In theory one can predict performance from a detailed knowledge of every part of the program and how this (the program) interacts with the specific hardware used. In practice, this is difficult to achieve and a good approach consists of establishing an approximate model for the amount of computation, memory, and other requirements of the program, and comparing this high-level model with the experience of actual performance.

In Table II, we present how the cpu time in computationally intensive steps scales with the two main parameters that define the problem size, $N$, the number of basis functions, and $n_{at}$, the number of atoms, as well as with the number of processors, $P$.

The SCF scales as $O(N^{2-4})/P$ with a serial overhead that is related to the diagonalization and other serial steps that are not parallelized and that scale, at most, as $O(N^3)$. The communication overhead is related to the global sum of the Fock operators performed in each SCF iteration.

The computation of the first- and second-derivative integral scales with the number of basis functions $N$ approximately in the same way as does the SCF, but it scales also with increasing number of atoms as $O(n_{at}^{1-2})$. The serial overhead of this step is a consequence of the computation of the first and second derivatives of the one-electron integrals, which is not parallelized. Again, there is communication overhead related to the global sum of the incomplete Hessian and of the Fock-like derivative matrices $\tilde{F}^a$.

The integral transformation is independent of the number of atoms and scales with $N$ as $O(N^{4-5})$ as our algorithm makes use of the sparsity in the first index integral transformation. There is a small serial overhead related to the extra manipulation of indices needed for the classification in buckets. The communication overhead comes from the exchange of the buckets in the first and third index transformations. It scales linearly with the number of two-electron integrals [this is the $O(N^{3-4})$ factor] and as $(P-1)/P$ with the number of processors. This is an important characteristic of this algorithm in view of its application to large processor counts. Scalability is an issue difficult to define precisely but in a broad sense a program can be said to be scalable if it will execute with acceptable efficiency on large number of processors. One important characteristic that must include any algorithm to be scalable is that the interprocessor communication should not increase or increase only very weakly with the number of processors. In our case this function is $(P-1)/P$, which grows slowly and it is upper-bounded by one. On this basis our program seems applicable to parallel computers with many processors.

The last computationally intensive step is the solution of the CPHF equations. In this step, the execution time grows approximately as $O(N^{3-4} \cdot n_{at})$, and the size of the CPHF equations grows with the number of two-electron integrals and the number of CPHF systems of equations is directly proportional to the number of atoms. There is small sequential overhead as a consequence of some steps that are not parallelized which grow, at most, as $O(N^2 \cdot n_{at}^2)$, and there is also communication overhead related to the global sum of the residual vectors at each iteration of the procedure.

## Performance

Timing results and speed-ups obtained in a cluster of HP 9000/735 workstations are presented in Figures 2 and 3 for $CH_3CH_2OCH_3$ with a TZP basis set as first test case (128 basis functions, 12 atoms). In all runs, the allocated memory per processor was limited to about 32 Mb.

**TABLE II.**
**Performance Model for the Program.**

| | Time parallel | Serial overhead | Communication overhead |
|---|---|---|---|
| SCF | $O(N^{2-4})/P$ | $O(N^3)$ | $O(N^2 \log_2 P)$ |
| JKDD | $O(n_{at}^{1-2}) \cdot O(N^{2-4})/P$ | $O(n_{at}^{1-2}) \cdot O(N^{1-2})$ | $O(n_{at}^2 \log_2 P) + O(n_{at} N^2 \log_2 P)$ |
| IJKL (std) | $O(N^{4-5})/P$ | | |
| IJKL (dir) | $[O(N^{4-5}) + O(N^{3-4})]/P$ | $O(N^{3-4})$ | $O(N^{3-4}) \cdot (P-1)/P$ |
| CPHF | $O(n_{at}) \cdot O(N^{4-5})/P$ | $O(N^2 n_{at}) + O(n_{at}^{1-2})$ | $O(n_{at} N^2 \log_2 P)$ |

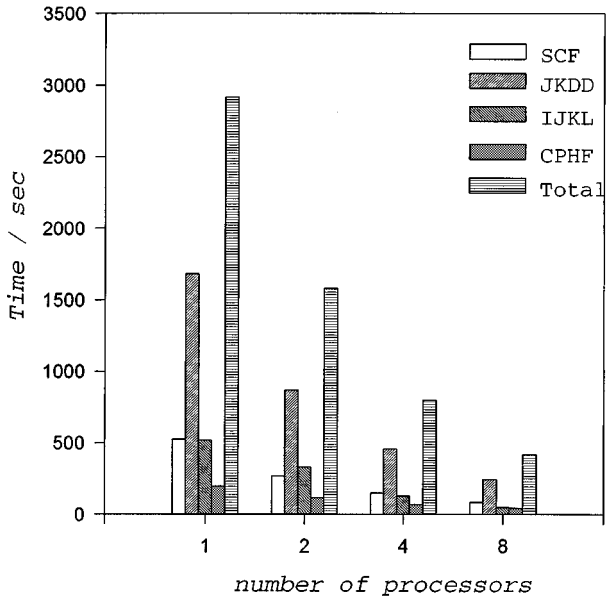


**FIGURE 2.** Timing results for $CH_3CH_2OCH_3$ sample system. SCF: SCF calculation; JKDD: AO integral first and second derivatives; IJKL: integral transformation; CPHF: resolution of the CPHF systems of equations.

In this sample, computation of the first and second derivatives of the integrals is the most time-consuming step, around 50% of the total time. The speed-up of this step is very high, as expected,[3b,m] going down to 6.91 when run on an eight-processor parallel computer. These data show

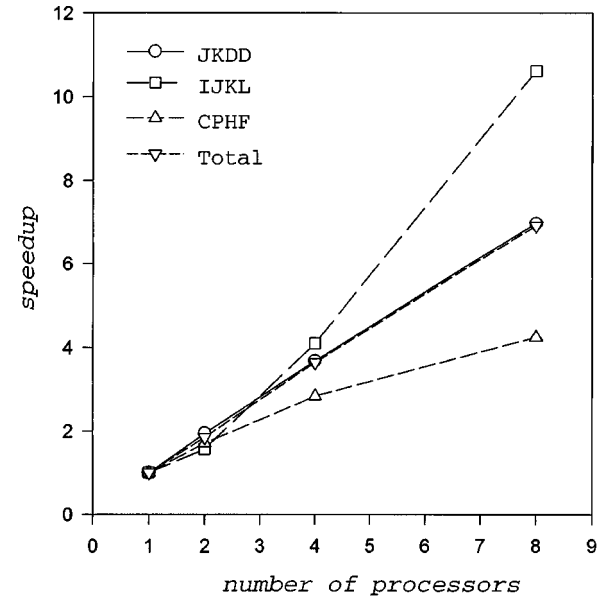


**FIGURE 3.** Speed-ups for $CH_3CH_2OCH_3$ sample system.

that, even for this small test sample, the effect of computation of the one-electron first- and second-derivative integrals, not parallelized, is small with respect to the computation of the two-electron derivative integrals.

The two-electron integral transformation shows an interesting increase in efficiency with increasing number of processors, going from a speed-up of 1.57 when run with two processors to 10.62 when run with eight processors. The relatively poor performance of the two processors run can be explained because of the time consumed by the PVM communication routines (39 seconds out of a total of 330 seconds for this step). By substracting the times consumed by the communication routines (that are not an inherent part of our algorithm and are related to buffer management in the PVM routines) we see that our algorithm is highly efficient with speed-ups ranging from 1.78 for the two-processor run, 4.96 for the four-processor run, and up to 12.37 for the eight-processor run. These superlinear speed-ups can be explained in terms of the increased available *global memory* accessible to the run when the number of processors increases. As discussed in a previous section, when main memory is limited the integral transformation program proceeds by batches, transforming as many occupied orbitals as possible on the first index transformation step. When the number of processors increases, the global memory accessible to the program increases reducing the number of batches and increasing the parallel efficiency of this step. It is expected that when the global memory is enough to transform all integrals in a single batch, further increases in number of processors will not result in further increases in superlinear speed-up, but linear speed-ups will continue with additional nodes. The increased efficiency of parallelization, even including the time consumed by the communication routines, is also indicative of the good scalability of the algorithm.

Finally, the resolution of the CPHF system of equations shows a very high efficiency of parallelization, as expected. Serial steps on this section of the program do not seem to affect its overall efficiency.

On average, very good overall efficiencies are seen, even at eight nodes, implying that even more nodes could be used even on this small test case. It is also interesting to note that the largest difference in cpu time between processors does not account for more than 5% of the total time, indicative of an excellent load balancing between processors.

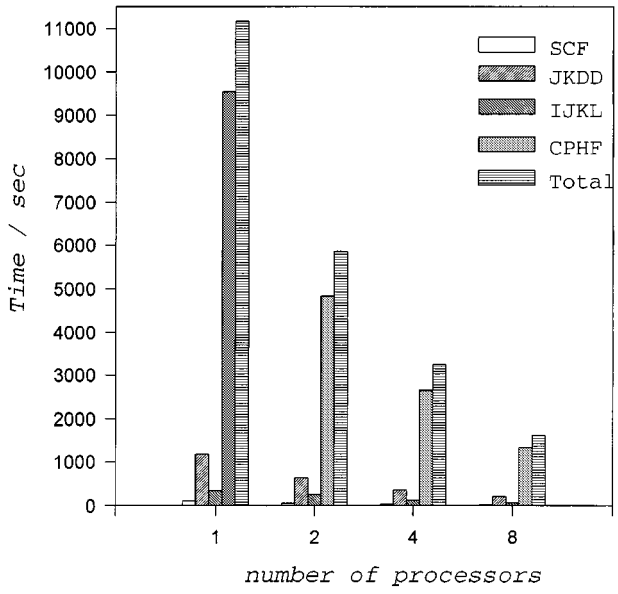


**FIGURE 4.** Timing results for 18-crown-6.

A different balance of the importance of the various steps of the calculation is observed when the number of atoms in the molecular system increases. As a second test case we used crown ether, the 18-crown-6, a system with 42 atoms and 114 basis functions. In Figures 4 and 5, timing results and speed-ups obtained in a cluster of IBM RS6000/590 computers are presented. One can note that, in this case, the resolution of the CPHF equations is the dominant step (around 80% of the total time) and thus determines the global speed-up of the run. The observed speed-up is nearly linear showing good efficiency of parallelization of the CPHF step.

For larger molecular systems, however, the available disk space will impose a limit on the size of the calculation that can be performed using standard techniques and a direct approach should be used instead. A parallel direct algorithm has also been implemented on which the two-electron AO repulsion integrals are never written to disk but recomputed when needed, thus avoiding disk space storage requirements at the expense of increased computational effort. However, MO integrals are still written to disk and, although they are distributed over the processors, the size of the MO integral file will impose a limit on the size of the molecular system that can be studied. A new algorithm in which no two-electron repulsion integrals (neither AO or MO integrals) need to be written to disk is being developed.

Timing and speed-up results obtained in a cluster of RS6000/590 computers for a direct Hessian calculation are presented in Figures 6 and 7, respectively. As a sample molecular system we have used, in this case, *p*-nitroaniline, with a DZP basis set (16 atoms, 180 basis functions). In this case we were unable to run this test on one processor due

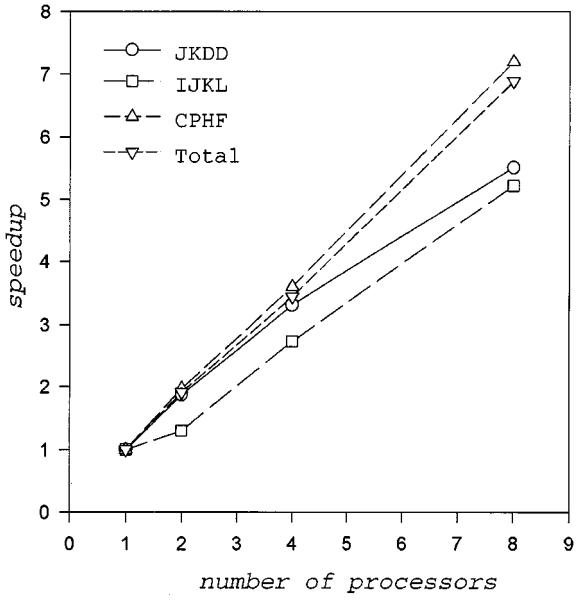


**FIGURE 5.** Speed-ups for 18-crown-6.

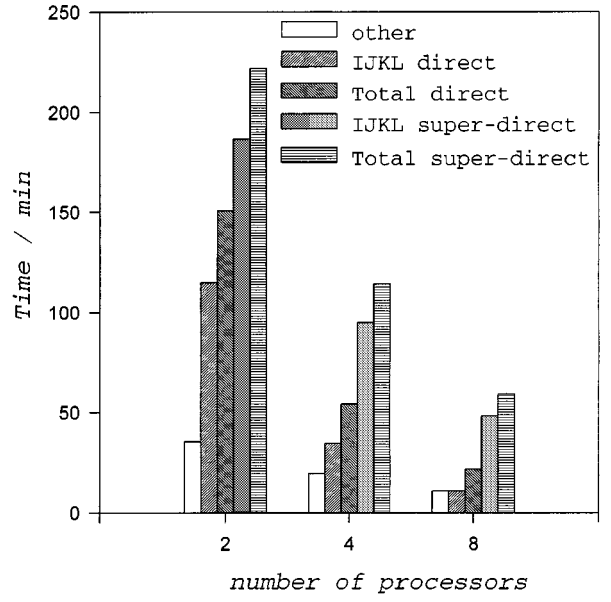


**FIGURE 6.** Timing results for *p*-nitroaniline. Direct calculation. Others: SCF + JKDD + CPHF; IJKL direct: integral transformation (direct algorithm); IJKL superdirect: integral transformation (superdirect algorithm).

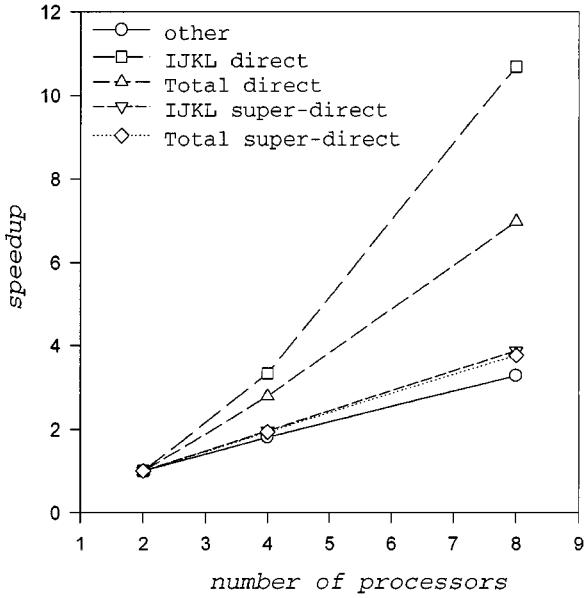


**FIGURE 7.** Speed-ups for *p*-nitroaniline. Direct calculation.

to the size of the system and the allocated memory per processor. In fact, this is a good example of enabling calculations on molecules that are otherwise too large to be handled sequentially. Although the examples used in this work are relatively small and can be run on single node machines, we can estimate that if one had, for example, 32 nodes with 128 Mb of memory each, a calculation with as many as 1600 basis functions could be run on a parallel computer with this configuration.

As the timings in Figure 6 show, the integral transformation is now the dominant step of the calculation with other steps (SCF, JKDD, CPHF) accounting for a small fraction of the total time. Two different algorithms have been implemented. The first one (direct) is basically the algorithm depicted in Figure 1 and has been discussed previously. The second one (super direct) attempts to eliminate the possible communication overhead in the first step of the integral transformation at the expense of computing all AO integrals on each thread. If the communication between nodes is slow it may be preferable to reduce the amount of data that needs to be exchanged even at the expense of increased computational effort (reduced communication overhead at the expense of increased sequential overhead).

Because the one-node times are unavailable, the data in Figure 7 are actually the speed-ups relative to the two-processor run. Large superlinear speed-ups are observed in the direct calculation reaching 10.7 for the eight-node run. In this run the integral transformation has been reduced to 50% of the time and it is expected that further increases in the number of nodes will reduce the observed speed-up of the total run. The relative speed-ups for the superdirect implementation are not as good as for the direct algorithm; however, they show good performance with nearly linear speed-ups.

## Concluding Remarks and Future Work

The results reported in this work show that, even for medium-sized calculations in parallel computers with a small number of processors, this parallel implementation of the analytic computation of the RHF nuclear Hessian performs extremely well. Important characteristics of the integral transformation code are its grain size, which allows for an adequate job distribution between processors; its limited interprocessor communication scheme, which results in an algorithm suitable for use in MPP computers; and the redistribution of the minimum required memory between the available nodes, increasing the range of applicability of the code to large molecular systems. All these characteristics and the observed performance indicate that the present algorithm can be of general use for the two-electron repulsion integral transformation in all cases in which such a task is needed. Further work is in progress to improve the integral transformation code and the AO integrals and derivatives code and to develop a new approach that will avoid computation and storage of the two-electron MO integrals.

## Acknowledgments

The authors thank the staff of the Centro Informático Científico de Andalucía, Convex Supercomputer SAE, and the Centro de Estudios y Experimentación de Obras Públicas for computational facilities. Jaime Oviedo thanks Convex Supercomputer SAE for a grant. This work was financially supported by the Dirección General Científica y Técnica (Project, No. PB92-0662) and by IBM Corp. (University Agreement MHVU4802).

## References

1. (a) P. Pulay, In *Ab Initio Methods in Quantum Chemistry—II*, K. P. Lawley, Ed., Wiley, Chichester, UK, and references

therein; (b) Y. Yamaguchi, Y. Osamura, J. D. Goddard, and H. F. Schaeffer, *A New Dimension to Quantum Chemistry*: *Analytic Derivative Methods in Ab Initio Molecular Electronic Structure Theory*, Oxford University Press, Oxford, UK, 1994.

2. R. A. Kendall, R. J. Harrison, R. J. Littlefield, and M. F. Guest, In K. P. Lipkowitz and D. B. Boyd, Eds., *Reviews in Computational Chemistry*, *Vol. VI*, VCH Publishers, New York, 1995.
3. (a) M. F. Guest, R. J. Harrison, J. H. van Lenthe, and L. C. H. van Corler, *Theor. Chim. Acta*, **71**, 117 (1987); (b) M. Dupuis and J. D. Watts, *Theor. Chim. Acta*, **71**, 91 (1987); (c) R. Ernenwein, M. M. Rohmer, and M. Bénard, *Comput. Phys. Commun.* **58**, 305 (1990); (d) M. M. Rohmer, J. Demuynck, M. Bénard, R. Wiest, C. Henriet, and R. Ernenwein, *Comp. Phys. Commun.* **60**, 127 (1990); (e) R. J. Harrison and R. A. Kendall, *Theor. Chim. Acta*, **79**, 337 (1991); (f) M. D. Cooper and I. H. Hillier, *J. Comp.-Aid. Mol. Des.*, **5**, 171 (1991); (g) S. Kindermann, E. Michel, and P. Otto, *J. Comput. Chem.*, **13**, 414 (1992); (h) H. P. Lüthi, J. E. Merts, M. Feyersen, and J. Almlöf, *J. Comput. Chem.*, **13**, 160 (1992); (i) M. Feyersen and R. A. Kendall, *Theor. Chim. Acta*, **84**, 289 (1993); (j) H. P. Lüthi and J. Almlöf, *Theor. Chim. Acta*, **84**, 443 (1993); (k) M. E. Colvin, C. L. Janssen, R. A. Whiteside, and C. H. Tong, *Theor. Chim. Acta*, **84**, 301 (1993); (l) S. Brode, H. Horn, M. Ehrig, D. Moldrup, J. E. Rice, and R. Ahlrichs, *J. Comput. Chem.*, **14**, 1142 (1993); (m) M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, N. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery Jr., *J. Comput. Chem.*, **14**, 1347 (1993); (n) T. R. Furlani and H. F. King, *J. Comput. Chem.*, **16**, 91 (1995).
4. (a) R. A. Whiteside, J. S. Blinkey, M. E. Colvin, and H. F. Schaefer III, *J. Chem. Phys.* **86**, 2185 (1987); (b) L. A. Covick and K. M. Sando, *J. Comput. Chem.*, **11**, 1151 (1990); (c) R. Wiest, J. Demuynck, M. Bénard, M. M. Rohmer, and R. Ernenwein, *Comp. Phys. Comm.* **62**, 107 (1991).
5. (a) J. D. Watts and M. Dupuis, *J. Comput. Chem.*, **9**, 158 (1988); (b) A. Márquez and M. Dupuis, *J. Comput. Chem.*, **16**, 395 (1995); (c) I. M. B. Nielsen and E. T. Seidl, *J. Comput. Chem.*, in press.
6. M. Schüler, T. Kovar, H. Lischka, R. Shepard, and R. J. Harrison, *Theor. Chim. Acta*, **84**, 489 (1993).
7. A. P. Rendell, T. J. Lee, and R. Lindh, *Chem. Phys. Lett.*, **194**, 84 (1993).
8. V. Sunderam, *Concurrency: Practice and Experience*, **2**, 315 (1990); (b) V. Sunderam, *Concurrent Computing with PVM*, Cluster Computing Workshop, Florida State University, 1992.
9. (a) H. F. King and A. Komornicki, In P. Jørgensen and J. Simons, Eds., *Geometrical Derivatives of Energy Surfaces and Molecular Properties*, D. Reidel, New York, 1986, p. 207; (b) H. F. King and A. Komornicki, *J. Chem. Phys.*, **84**, 5645 (1986).
10. The coefficient for the first term in eq. (4) is incorrectly displayed in ref. 9a. See, for example, ref. 1b.
11. S. Saebo and J. Almlöf, *Chem. Phys. Lett.*, **154**, 83 (1989).
12. J. A. Pople, R. Krishnan, H. B. Schlegel, and J. S. Blinkey, *Int. J. Quantum Chem., Quantum Chem. Symp.*, **13** 225 (1979).